南华大学

机电类创新训练中心

数字闹钟的设计

1.1 数字闹钟的设计

1.1.1 系统的设计要求

▲ 设计一个24小时的数字闹钟，它包括以下几个组成部分：

①显示屏：由4个七段数码管组成，显示时间(闹钟)(时：分)

②数字键'0'-'9'：用于输入新的时间或新的闹钟时间；

③TIME(时间)键：用于确定新的时间设置；

④ALARM(闹钟)键：用于确定新的闹钟时间设置，或显示已设置的闹钟时间；

⑤扬声器：在当前时钟时间与闹钟时间相同时，发出蜂鸣声。

该数字闹钟的面板图:

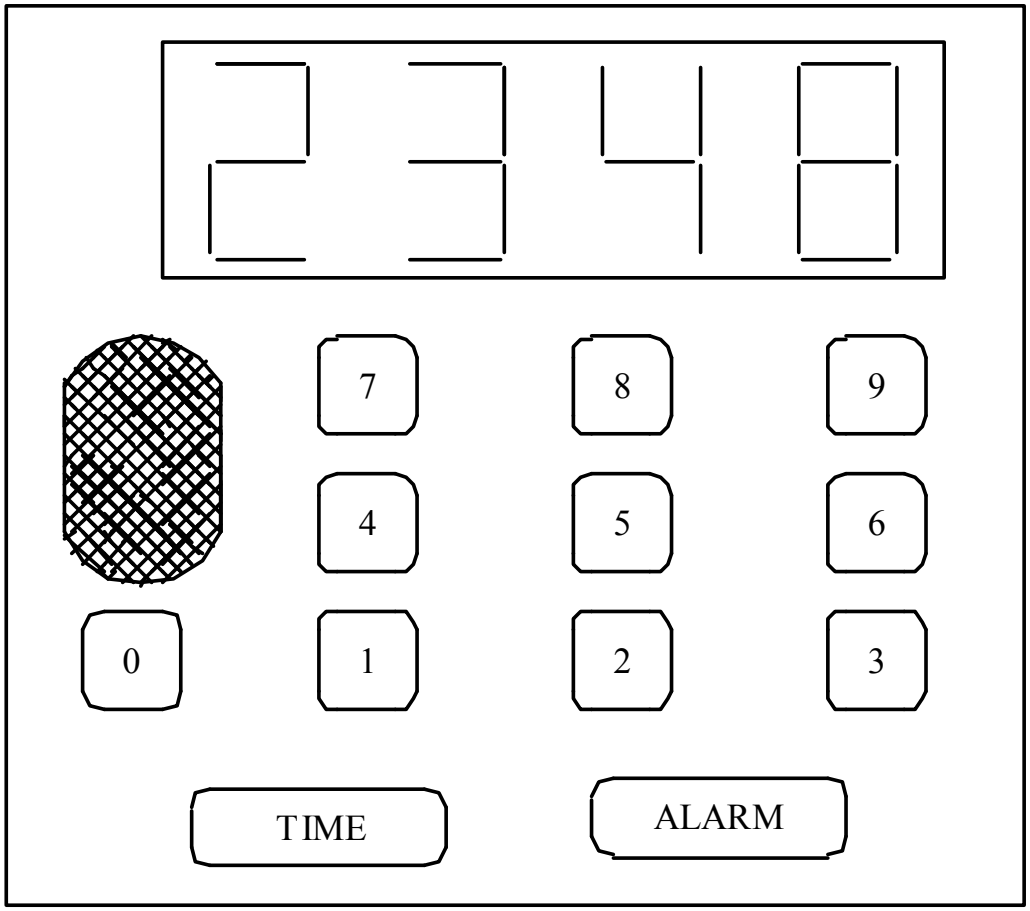


图1.1数字闹钟的面板图

▲该数字闹钟的功能：

(1) **计时功能：**每隔一分钟计时一次，并显示当前时间。

(2) **闹钟功能：**当前时间与闹钟时间相同，扬声器发出蜂鸣声。

(3) **设置新的计时时间：**用户用数字键‘0’～‘9’输入新的时间，然后按 “TIME”键确认。在输入过程中，输入数字在显示屏上从右到左依次显示。例如，用户要设置新的时间12：34，则按顺序输入“1”，“2”，“3”，“4”键，与之对应，显示屏上依次显示的信息为：“1”，“12”，“123”，“1234”。如果用户在输入任意几个数字后较长时间内，例如5s，没有按任何键，则数字闹钟恢复到正常的计时显示状态。

(4) **设置新的闹钟时间：**用户用数字键“0”-“9”输入新的时间，然后按“ALARM”键确认。过程与(3)类似。

(5) **显示所设置的闹钟时间：**在正常计时显示状态下，用户直接按下“ALARM”键，则已设置的闹钟时间将显示在显示屏上。

1.1.2 系统的总体设计

1. 整体组装端口说明

将该数字闹钟命名为ALARM_CLOCK，根据该数字闹钟的设计要求，其外部端口如图7.2所示。

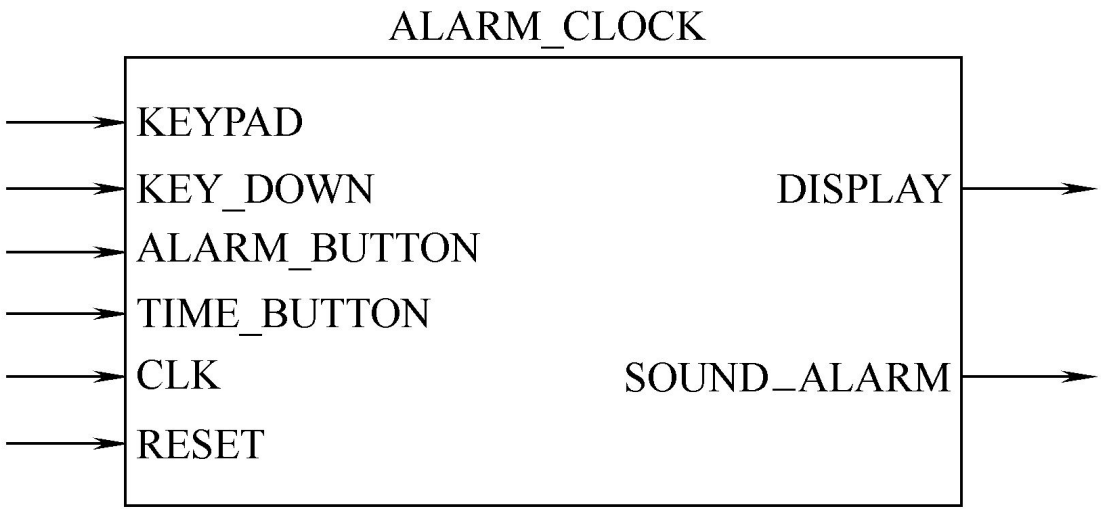
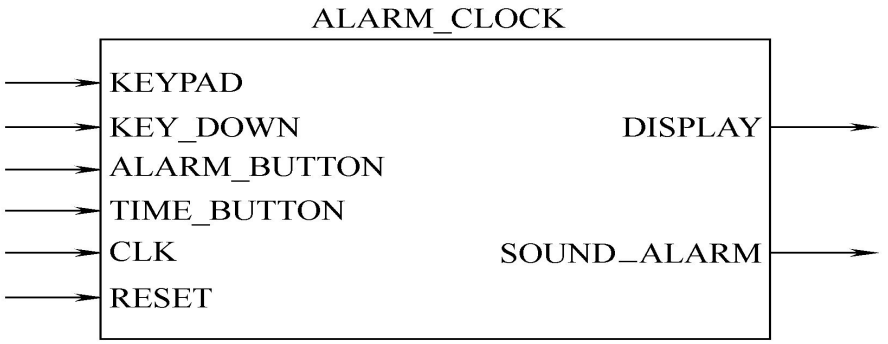


图1.2 数字闹钟的外部端口

输入端口：



① **CLK：** 外部时钟信号。

② **RESET：** 外部复位信号。

③ **KEYPAD：** 10位信号，若其中某一位为高电平，则表示用户按下了相应下标的数字键。例如，若KEYPAD(5) = ‘1’，表示用户按下了数字键“5”。本设计不考虑用户同时按下多个数字键的情况，所以任意时刻KEYPAD中只能有一位为‘1’。

④ **KEY_DOWN：** (KEY_DOWN= ‘1’)，表示用户按下某一数字键。

⑤ **ALARM_BUTTON：** 为高电平时，表示用户按下ALARM键。

⑥ **TIME_BUTTON：** 为高电平时，表示用户按下TIME键。

输出端口：

① **DISPLAY：** 实际上输出驱动4个七段数码显示管的数据，用于显示时间，如12:20。

② **SOUND_ALARM：** 用于控制扬声器发声，当SOUND_ALARM='1'时，扬声器发出蜂鸣，表示到了设定的闹钟时间。

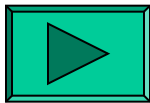
2. 各组成模块的功能：

根据系统的设计要求，整个系统分为7个模块：

(1) **闹钟控制器(ALARM_CONTROLLER)：** 是整个系统正常有序工作的核心，按设计要求产生相应的控制逻辑，以控制其他各部分的工作。

(2) **译码器(DECODER)：** 可将KEYPAD信号转换为0～9的整型数，以直观地表示和处理用户输入的数字。

(3) **键盘缓冲器(KEY_BUFFER)：** 是一个移位寄存器，暂存用户键入的数字，并且实现用户键入数字在显示器上从右到左的依次显示。这里需要注意的是，由图7.3可以看出，KEY_BUFFER的时钟端连接的是外部KEY_DOWN 信号。这表示用户每输入一个数字，KEY_BUFFER移位一次。

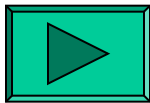


(4) **分频器(FQ_DIVIDER)：** 将较高速的外部时钟频率分频成每分钟一次的时钟频率，以便进行时钟计数。

(5) **时间计数器(ALARM_COUNTER)：** 实际上是一个异步复位、异步置数的累加器，通常情况下进行时钟累加计数，必要时可置入新的时钟值，然后从该值开始新的计数。

(6) **闹钟寄存器(ALARM_REG)：** 用于保存用户设置的闹钟时间，是一个异步复位寄存器。

(7) **显示驱动器(DISPLAY_DRIVER)：** 根据需要显示当前时间、用户设置的闹钟时间或用户通过键盘输入的新的时间，同时判断当前时间是否已到了闹钟时间，实际上是一个多路选择器加比较器。



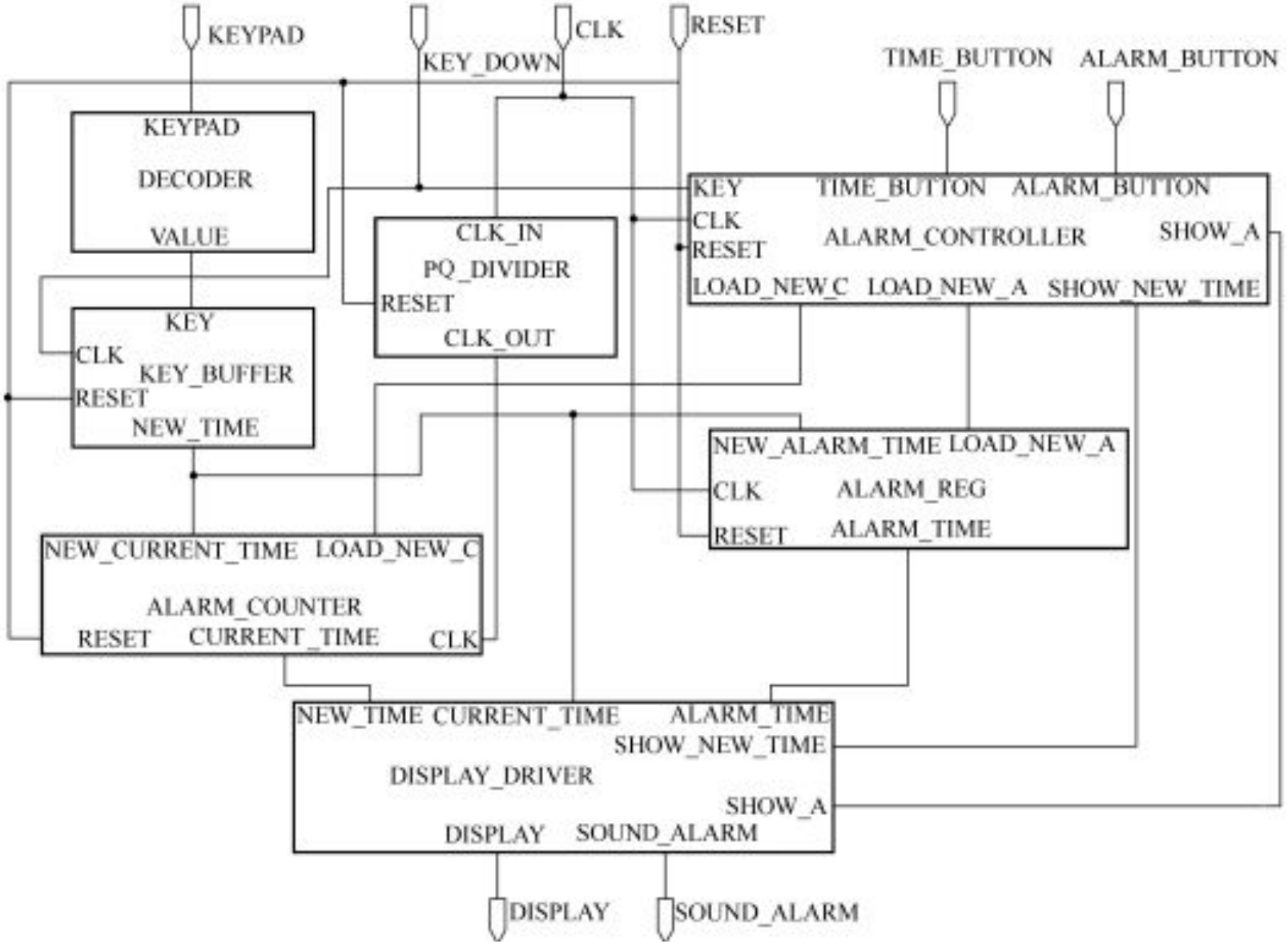

KEYPAD
KEY_DOWN
CLK
RESET
TIME_BUTTON
ALARM_BUTTON
KEYPAD
DECODER
VALUE
CLK_IN
PQ_DIVIDER
RESET
CLK_OUT
KEY
TIME_BUTTON
ALARM_BUTTON
CLK
RESET
ALARM_CONTROLLER
SHOW_A
LOAD_NEW_C
LOAD_NEW_A
SHOW_NEW_TIME
KEY
CLK
KEY_BUFFER
RESET
NEW_TIME
NEW_ALARM_TIME
LOAD_NEW_A
CLK
ALARM_REG
RESET
ALARM_TIME
NEW_CURRENT_TIME
LOAD_NEW_C
ALARM_COUNTER
RESET
CURRENT_TIME
CLK
NEW_TIME
CURRENT_TIME
ALARM_TIME
SHOW_NEW_TIME
DISPLAY_DRIVER
SHOW_A
DISPLAY
SOUND_ALARM
DISPLAY
SOUND_ALARM

图1.3 总体结构

1.1.3 单元模块设计：

1、闹钟控制器的设计

(1)模块功能及端口定义：

控制器命名为ALARM_CONTROLLER，其外部如图7.4所示。

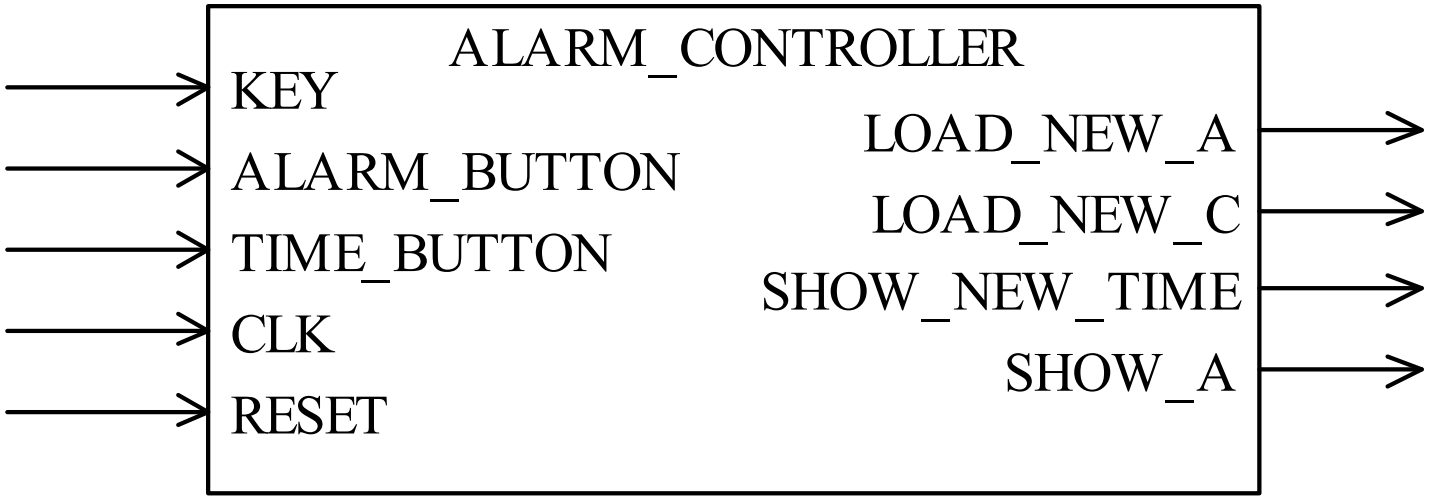
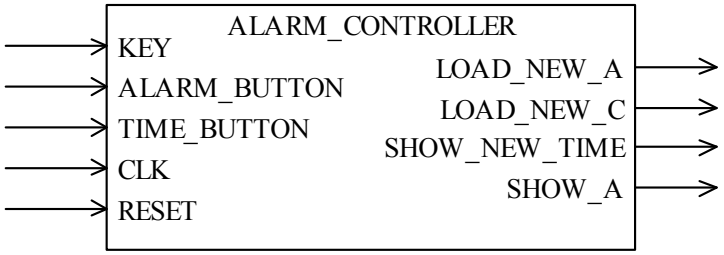


图1.4 控制器的外部端口

输入端口：



(1) CLK：外部时钟信号。

(2) RESET：复位信号。

(3) KEY：用户按下数字键(“0”～“9”)标志，高电平有效。

(4) ALARM_BUTTON：用户按下“ALARM”键标志，高电平有效。

(5) TIME_BUTTON：用户按下“TIME”键标志，高电平有效。

输出端口：

(1) LOAD_NEW_A: 高电平有效，控制(闹钟时间寄存器)加载新的闹钟时间值。

(2) LOAD_NEW_C: 高电平有效，控制(时钟计数器)设置新的时间值。

(3) SHOW_NEW_TIME：

SHOW_NEW_TIME=1：控制(七段数码显示电路)显示新的时间值，即用户通过数字键输入的时间；

SHOW_NEW_TIME=0：当SHOW_A=1，控制显示闹钟时间，

当SHOW_A=0，控制显示当前时间。

(2)状态定义： 控制器的功能可以通过有限状态机(FSM)的方式来实现。根据设计要求及端口设置，需要5个状态来实现：

S0：表示电路初态即正常时钟计数状态，完成前面设计功能（1）的工作。

S1：接收键盘输入状态。在状态S0时用户按下数字键后进入此状态。在此状态下，显示屏上显示的是用户键入的数字。

S2：设置新的闹钟时间。在状态S1时用户按下ALARM 键后进入此状态。

S3：设置新的计时时间。在状态S1时用户按下TIME 键后进入此状态。

S4：显示闹钟时间。在状态S0时用户直接按下ALARM 键后进入此状态。在此状态下，显示屏上显示的是所设置的闹钟时间。注意：在此状态下，用户按下ALARM 键后，显示屏上保持显示闹钟时间，经过一段时间以后，再返回状态S0显示数字闹钟时间。

相应的状态转换及控制如表1.1所示。

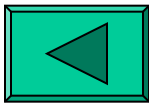
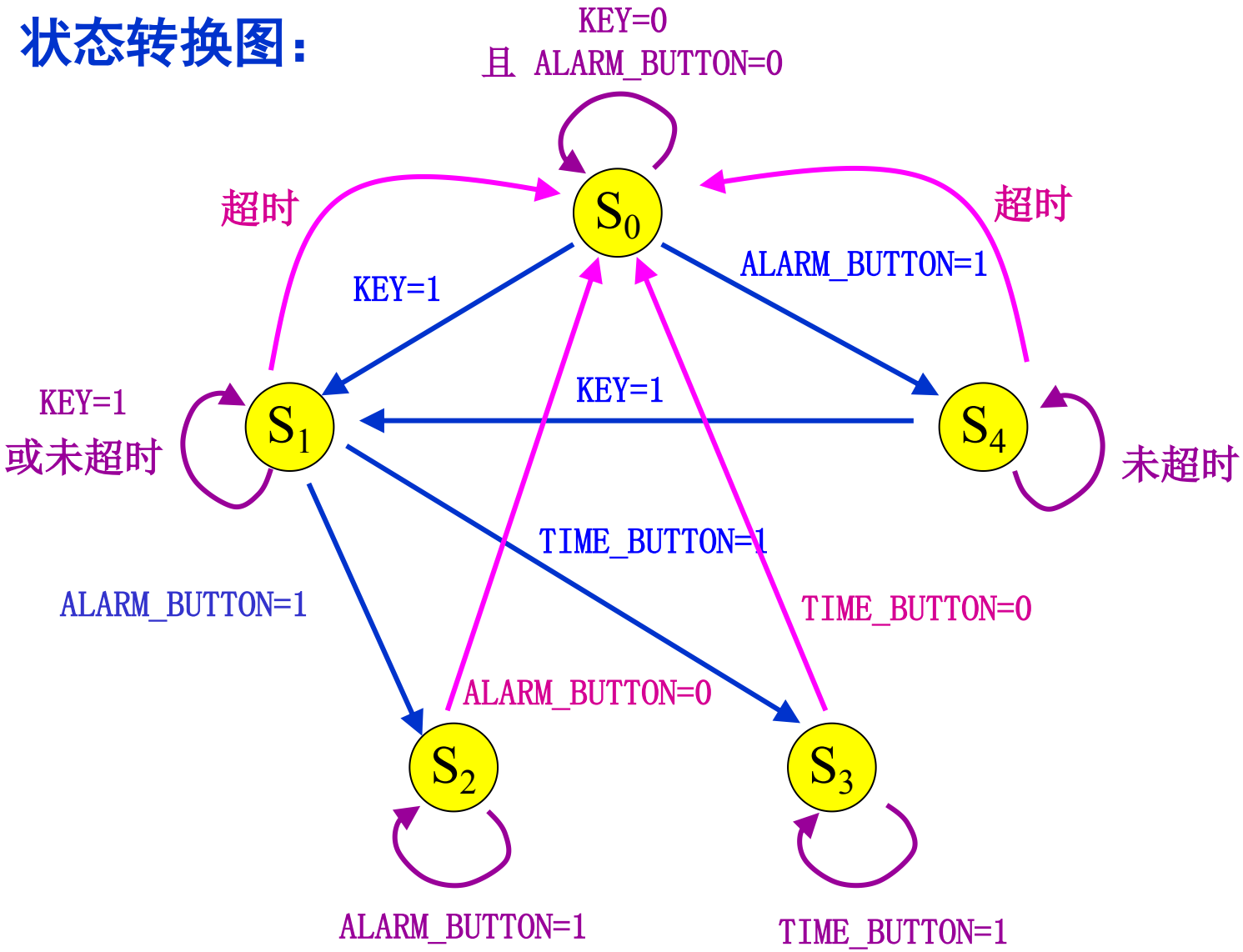


表1.1 控制器状态转换及控制输出表

当前状态	控制输入(条件)		下一状态	控制输出(动作)
S0	KEY = ‘1’		S1	SHOW_NEW_TIME <= ‘1’
	ALARM_BUTTON = ‘1’		S4	SHOW_A <= ‘1’
	否则		S0	--
S1	KEY = ‘1’		S1	SHOW_NEW_TIME <= ‘1’
	ALARM_BUTTON = ‘1’		S2	LOAD_NEW_A <= ‘1’
	TIME_BUTTON = ‘1’		S3	LOAD_NEW_C <= ‘1’
	否则(超时)	否	S1	SHOW_NEW_TIME <= ‘1’， “超时”判断处理
		是	S0	--
S2	ALARM_BUTTON = ‘1’		S2	LOAD_NEW_A <= ‘1’
	否则		S0	--
S3	TIME_BUTTON = ‘1’		S3	LOAD_NEW_C <= ‘1’
	否则		S0	--
S4	ALARM_BUTTON = ‘1’		S4	SHOW_A <= ‘1’
	否则(超时)	否	S4	SHOW_A <= ‘1’， “超时”判断处理
		是	S0	--

表7.1中没有显式说明的控制信号赋值，表示信号的值为零。例如在状态S0，当信号KEY =‘1’时，SHOW_NEW_TIME信号的赋值为‘1’，而其他信号LOAD_NEW_A，LOAD_NEW_C和SHOW_A的值此时都赋为‘0’。

(3) 状态转换图：


KEY=0
且 ALARM_BUTTON=0
超时
超时
S0
KEY=1
ALARM_BUTTON=1
KEY=1
KEY=1
或未超时
S1
S4
未超时
TIME_BUTTON=1
ALARM_BUTTON=1
TIME_BUTTON=0
ALARM_BUTTON=0
S2
S3
ALARM_BUTTON=1
TIME_BUTTON=1

(4) VHDL源程序：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.P_ALARM.ALL;
ENTITY ALARM_CONTROLLER IS
    PORT(    RESET ： IN STD_LOGIC;
                CLK ： IN STD_LOGIC;
                KEY ： IN STD_LOGIC;
            ALARM_BUTTON ： IN STD_LOGIC;
            TIME_BUTTON ： IN STD_LOGIC;
            LOAD_NEW_A ： OUT STD_LOGIC;
            LOAD_NEW_C ： OUT STD_LOGIC;
        SHOW_NEW_TIME ： OUT STD_LOGIC;
                SHOW_A  ： OUT STD_LOGIC);
END ALARM_CONTROLLER;
```

```
ARCHITECTURE ART OF ALARM_CONTROLLER IS
    TYPE    T_STATE IS (S0，  S1，  S2，  S3，S4)；
    CONSTANT   KEY_TIMEOUT ：T_SHORT ：= 500；
    CONSTANT   SHOW_ALARM_TIMEOUT ：  T_SHORT ：= 500；
    SIGNAL  CURR_STATE ：  T_STATE；
    SIGNAL  NEXT_STATE ：  T_STATE；
    SIGNAL  COUNTER_K  ：  T_SHORT；
    SIGNAL  ENABLE_COUNT_K ：  STD_LOGIC；
    SIGNAL  COUNT_K_END ：  STD_LOGIC；
    SIGNAL  COUNTER_A  ：  T_SHORT；
    SIGNAL  ENABLE_COUNT_A ：  STD_LOGIC；
    SIGNAL  COUNT_A_END ：  STD_LOGIC；
```

设置值

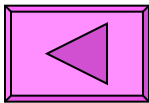
常数名

数据类型

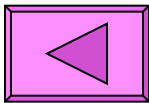
```
BEGIN
    PROCESS(CLK，RESET)
    BEGIN
        IF RESET ='1' THEN
            CURR_STATE <= S0；          --异步复位
        ELSIF RISING_EDGE(CLK) THEN
            CURR_STATE <= NEXT_STATE；
        END IF；                        --状态转换
    END PROCESS；
```

```
PROCESS(KEY，ALARM_BUTTON，TIME_BUTTON，
            CURR_STATE，  COUNT_A_END，COUNT_K_END)
BEGIN
    NEXT_STATE    <= CURR_STATE；
    LOAD_NEW_A    <= '0'；
    LOAD_NEW_C    <= '0'；
    SHOW_A        <= '0'；
    SHOW_NEW_TIME  <= '0'；
    ENABLE_COUNT_K <= '0'；
    ENABLE_COUNT_A <= ‘0’；     - - 设初值，初始化
```

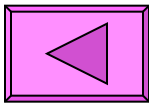
```
CASE CURR_STATE IS
  WHEN S0 =>
      IF (KEY = '1') THEN
          NEXT_STATE <= S1;
          SHOW_NEW_TIME <= ‘1’;      -- 控制显示键入的时间
      ELSIF (ALARM_BUTTON = '1') THEN
          NEXT_STATE <= S4;
          SHOW_A <= ‘1’;                -- 控制显示闹钟时间
      ELSE
          NEXT_STATE <= S0;
      END IF;
```

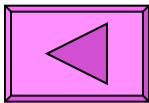
```
WHEN S1 =>
        IF (KEY = '1') THEN
              NEXT_STATE <= S1;
        ELSIF (ALARM_BUTTON = '1') THEN
              NEXT_STATE <= S2;
              LOAD_NEW_A <= ‘1’;   - - 控制设定新的闹钟时间
        ELSIF (TIME_BUTTON = '1') THEN
              NEXT_STATE <= S3;
              LOAD_NEW_C <= ‘1’;   - - 控制设定新的计时时间
        ELSE
              IF (COUNT_K_END = '1') THEN
                  NEXT_STATE <= S0;
              ELSE
                  NEXT_STATE <= S1;   - - 超时判断
              END IF;
          ENABLE_COUNT_K <= '1';  - - 打开超时判断定时器
        END IF;
        SHOW_NEW_TIME  <= ‘1’;   - - 控制显示键入的时间
```



```
WHEN S2 =>
      IF (ALARM_BUTTON = '1') THEN
            NEXT_STATE <= S2;
            LOAD_NEW_A <= ‘1’;   - - 控制设定新的闹钟时间
      ELSE
            NEXT_STATE <= S0;
      END IF;
WHEN S3 =>
      IF (TIME_BUTTON = '1') THEN
            NEXT_STATE <= S3;
            LOAD_NEW_C <= ‘1’;   - - 控制设定新的计时时间
      ELSE
            NEXT_STATE <= S0
      END IF;
```

```
    WHEN S4  =>
        IF (KEY = '1') THEN
            NEXT_STATE <= S1;
        ELSE
            NEXT_STATE <= S4;
          IF (COUNT_A_END = '1') THEN
            NEXT_STATE <= S0;        --超时返回S0
          ELSE
            NEXT_STATE <= S4;
            SHOW_A <= '1';           --显示闹钟时间
          END IF;
            ENABLE_COUNT_A <= '1';
        END IF;
     WHEN OTHERS =>
        NULL;
    END CASE;
END PROCESS;
```

```
COUNT_KEY：  PROCESS(ENABLE_COUNT_K，  CLK)
BEGIN
        IF (ENABLE_COUNT_K = '0') THEN
            COUNTER_K   <= '0';
            COUNT_K_END <= '0';
        ELSIF (RISING_EDGE(CLK)) THEN
            IF (COUNTER_K >= KEY_TIMEOUT) THEN
                COUNT_K_END <= '1';
            ELSE
                COUNTER_K <= COUNTER_K + 1;
            END IF;
        END IF;
END PROCESS;                          - - 按键间隔定时程序
```

```
COUNT_ALARM ： PROCESS(ENABLE_COUNT_A， CLK)
BEGIN
  IF (ENABLE_COUNT_A = '0') THEN
      COUNTER_A  <= 0；
      COUNT_A_END <= '0'；
  ELSIF RISING_EDGE(CLK) THEN
    IF (COUNTER_A >= SHOW_ALARM_TIMEOUT) THEN
        COUNT_A_END <= '1'；
    ELSE
        COUNTER_A <= COUNTER_A + 1；
    END IF；
   END IF；
END PROCESS；        --“显示闹钟时间”定时程序
END ART；
```

(5) 闹钟控制器的仿真结果：

2. 键盘译码器的设计

(1) 模块功能及端口定义：

本模块的功能是将每次按下闹钟系统的数字键盘后产生的一个数字所对应的10位二进制数据信号转换为1位十进制整数信号，以作为小时、分钟计数的4个数字之一。

如图7.5所示：

KEYPAD： 输入端口，接收10位二进制数据信号；

VALUE： 输出端口，输出相应的1位十进制整数信号。

输入数据与输出数据的译码关系见表7.2。

3. 键盘缓冲器(预置寄存器)的设计

(1) 模块功能及端口定义：

该模块的功能是在CLK端口输入信号的上升沿同步下，将KEY端口的输入信号移入NEW_TIME 端口的输出信号最低位，原有信息依次向左移，最高位信息丢失。

输入端口：

① **CLK：** 把用户按下某一数字键信号（KEY_DOWN）作为时钟信号。

② **RESET：** 外部异步复位信号。

③ **KEY：** 键盘缓冲器的输出数字数值。

输出端口：

NEW_TIME： 输出信号新的时间数值。

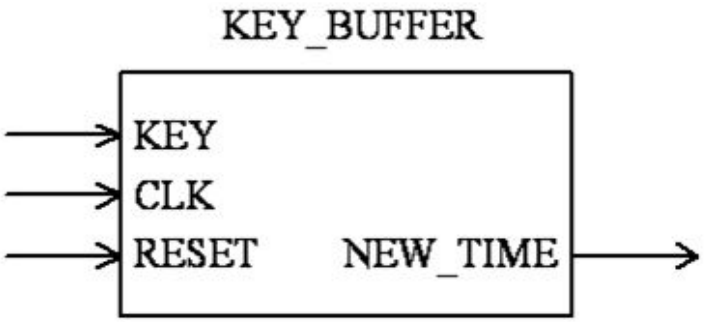


图1.6 键盘缓冲器模块示意图

4. 闹钟寄存器的设计

提示：

闹钟系统的闹钟时间由闹钟寄存器保存和传递，而当前时间由时间计数器保存、传递并按分钟累加推进。这两个组件的功能和设计描述比较相似，它们之间的区别主要在于自动累加功能的有无和控制信号的优先作用次序。

4. 闹钟寄存器的设计

(1) 模块功能及端口定义：

该模块的功能是在时钟上升沿同步下，根据LOAD_NEW_A端口的输入信号控制ALARM _TIME端口的输出，当控制信号有效(高电平)时，把NEW _ALARM _TIME端口的输入信号值输出。

输入端口：

① **CLK：** 外部时钟信号。

② **LOAD_NEW_A：** 新闹钟时间设定控制端。

③ **NEW _ALARM _TIME：** 新设定的闹钟时间值。

④ **RESET：** 外部异步复位信号。

输出端口：

ALARM_TIME： 输出闹钟时间。

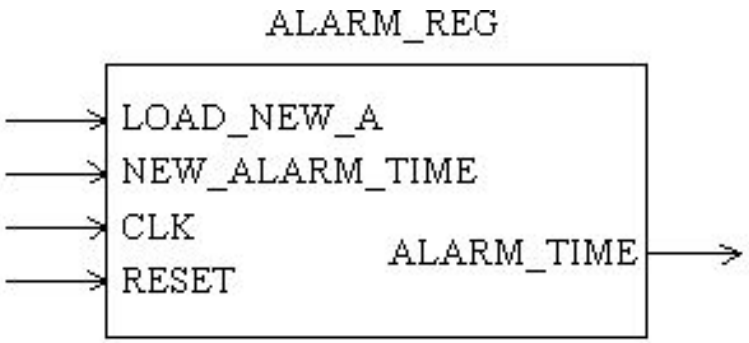


图1.7 闹钟寄存器示意图

5. 时间计数器的设计

(1) 模块功能及端口定义：

该模块的功能是当RESET输入端口为高电平时，对CURRENT_TIME输出端口清零；当LOAD_NEW_C输入端口为高电平时，将NEW_CURRENT_TIME输入端口的信号输出给CURRENT _TIME端口。RESET端口的控制优先于LOAD_NEW_C端口。当这两个控制信号都无效时，在时钟上升沿同步下，对CURRENT _TIME端口输出信号累加1，并根据小时、分钟的规律处理进位。

输入端口：

① **CLK：** 把外部时钟分频后的分脉冲作为时钟信号。

② **RESET：** 外部异步复位信号。

③ **LOAD_NEW_C：** 新计时时间设定控制端。

④ **NEW _CURRENT _TIME：** 新设定的计时时间值。

输出端口：

CURRENT_TIME： 输出当前计时时间。

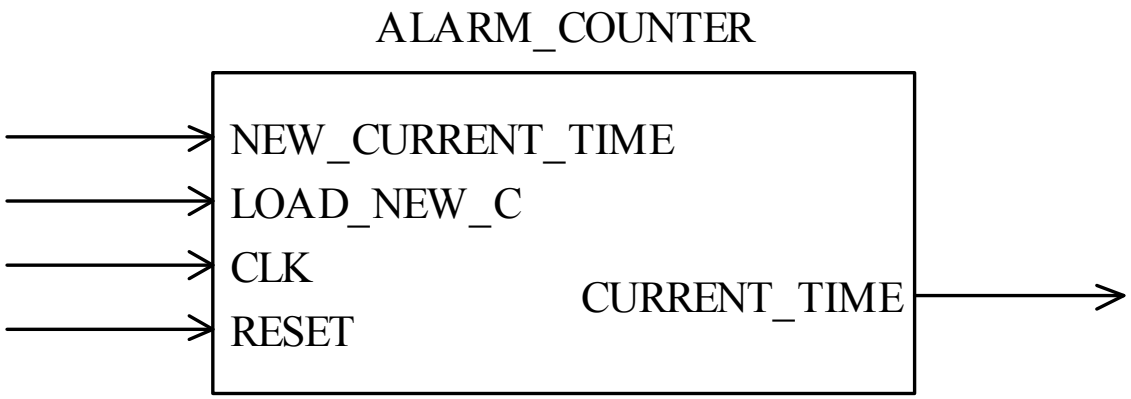


图1.8 时间计数器模块的示意图

6. 分频器的设计

(1) 模块功能及端口定义：

该模块的功能是将CLK_IN端口输入的时钟信号分频后送给CLK_OUT端口。

输入端口：

① **CLK**：外部时钟信号。

② **RESET**：外部异步复位信号。

输出端口：

CLK_OUT：输出分脉冲。

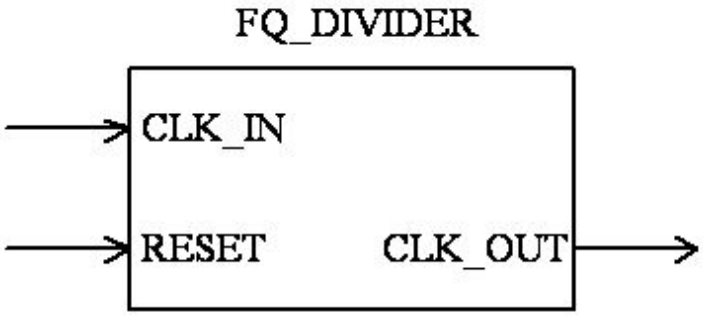


图1.9 分频器示意图

7. 显示驱动器的设计

(1) 模块功能及端口定义：

该模块的功能是：

① 输入信号SHOW_NEW_TIME=1时，DISPIDY输出用户键入时间NEW_TIME 的4个七段数码显示器的驱动数据。

② 输入信号SHOW_NEW_TIME=0时：

当SHOW_A=1时，DISPIDY输出闹钟时间ALARM_TIME的4个七段数码显示器的驱动数据。

当SHOW_A=0时，DISPIDY输出当前时间CURRENT_TIME的4个七段数码显示器的驱动数据。

③当输入信号ALARM_TIME = CURRENT_TIME时，SOUND_ALARM端口的输出信号有效(高电平)，反之无效。

输入端口：

(1) SHOW_NEW_TIME：键入时间显示控制端。

(2) SHOW_A：闹钟时间和当前时间选择显示控制端。

(3) NEW_TIME： 当前用户键入时间数据。

(4) ALARM_TIME：闹钟时间数据。

(5) CURRENT_TIME：当前计时时间数据。

输出端口：

(1) DISPYAY: 输出4个七段数码显示器的驱动数据。

(2) SOUND_ALARM: 高电平有效，闹钟报警输出。

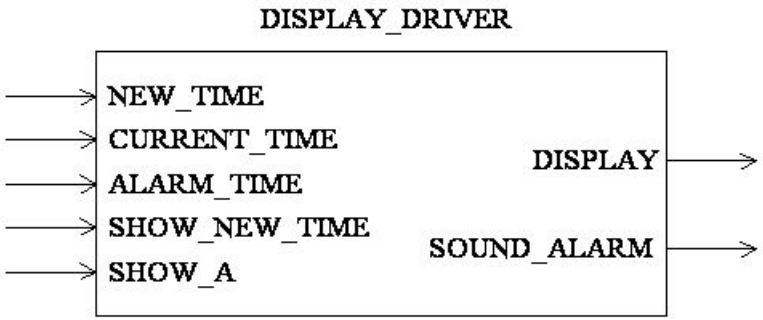
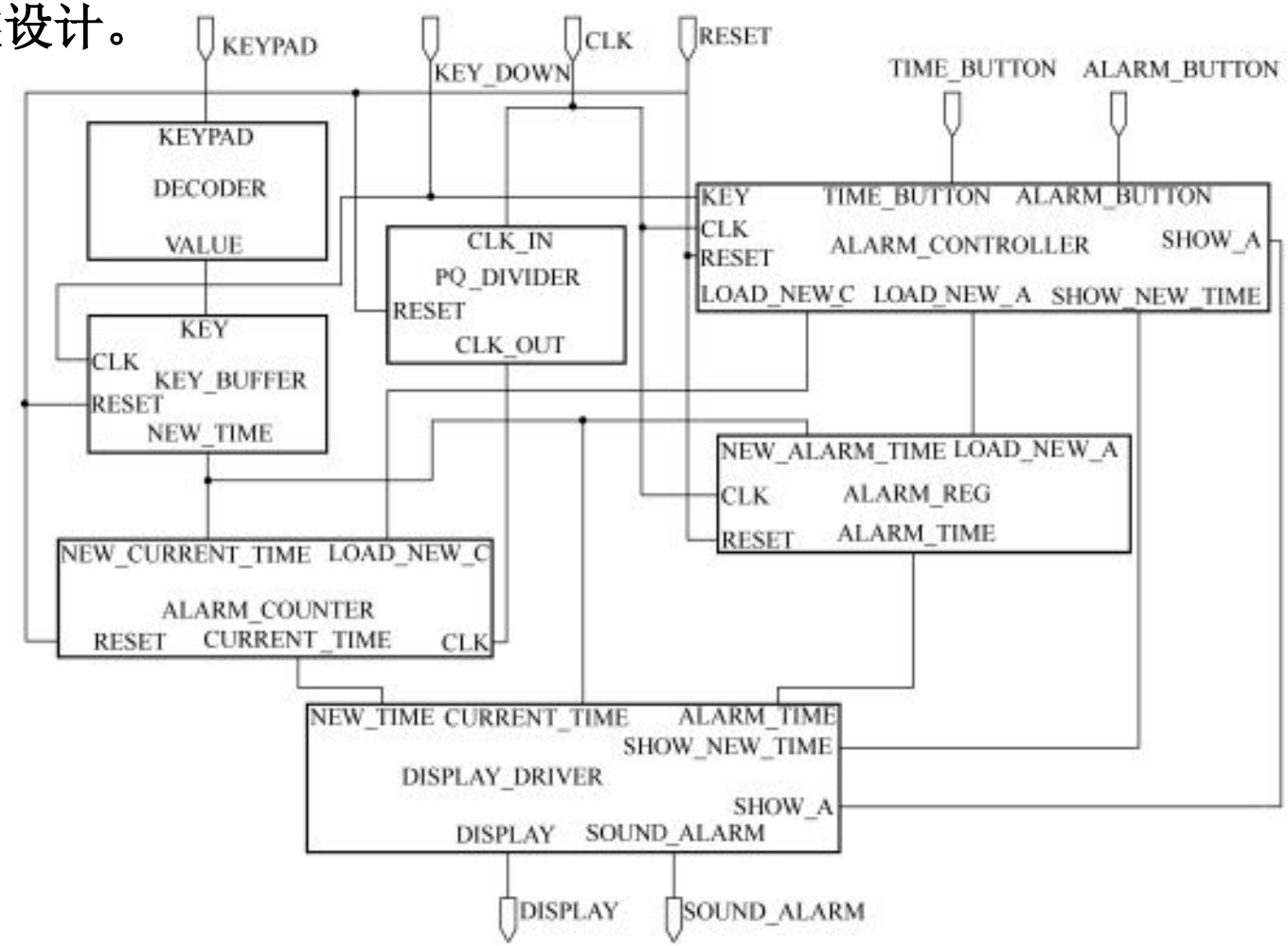


图1.10 显示驱动器示意

1.1.4 闹钟系统的整体组装

前面已经完成了数字闹钟各部分的设计，把各部分组装起来形成完整的总体设计，如图所示。可以用原理图或VHDL文本输入方式完成数字闹钟的顶层总装设计。



(2) 顶层总装设计VHDL源程序：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.P_ALARM.ALL;
ENTITY ALARM_CLOCK IS
  PORT ( KEYPAD ： IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
         KEY_DOWN ： IN  STD_LOGIC;
         ALARM_BUTTON：IN  STD_LOGIC;
         TIME_BUTTON ： IN  STD_LOGIC;
         CLK  ： IN  STD_LOGIC;
         RESET ： IN  STD_LOGIC;
         DISPLAY ： OUT T_DISPLAY;
         SOUND_ALARM ： OUT STD_LOGIC);
END ALARM_CLOCK;
```

```
ARCHITECTURE ART OF ALARM_CLOCK IS
  COMPONENT DECODER  -- 待调用元件端口定义
     PORT(KEYPAD：IN STD_LOGIC_VECTOR(9 DOWNTO 0)；
                VALUE ：OUT T_DIGITAL)；
     END COMPONENT；
  COMPONENT KEY_BUFFER  -- 待调用元件端口定义
      . . .
  COMPONENT ALARM_COUNTER  --待调用元件端口定义
    . . .
  COMPONENT ALARM_REG  -- 待调用元件端口定义
    . . .
  COMPONENT ALARM_CONTROLLER  -- 待调用元件端口定义
      . . .
```

```
COMPONENT DISPLAY_DRIVER  -- 待调用元件端口定义
  . . .
COMPONENT FQ_DIVIDER      -- 待调用元件端口定义
    . . .
SIGNAL INNER_KEY ： T_DIGITAL;
SIGNAL INNER_TIME ： T_CLOCK_TIME;
SIGNAL INNER_TIME_C ： T_CLOCK_TIME;
SIGNAL INNER_TIME_A ： T_CLOCK_TIME;
SIGNAL INNER_L_C ： STD_LOGIC;
SIGNAL INNER_L_A ： STD_LOGIC;
SIGNAL INNER_S_A ： STD_LOGIC;
SIGNAL INNER_S_N ： STD_LOGIC;
SIGNAL INNER_SEC_CLK ： STD_LOGIC;
```

```
FOR ALL：  DECODER  USE  ENTITY  WORK.DECODER(ART)；
FOR ALL：  KEY_BUFFER  USE  ENTITY   WORK.KEY_BUFFER(ART)；
FOR ALL：  ALARM_COUNTER  USE  ENTITY
                WORK.ALARM_COUNTER(ART)；
FOR ALL：  ALARM_REG  USE  ENTITY WORK.ALARM_REG(ART)；
FOR ALL：  ALARM_CONTROLLER  USE  ENTITY
                WORK.ALARM_CONTROLLER(ART)；
FOR ALL：  DISPLAY_DRIVER  USE  ENTITY
                WORK.DISPLAY_DRIVER(ART)；
F OR ALL：  FQ_DIVIDER  USE  ENTITY  WORK.FQ_DIVIDER(ART)；
```

```
BEGIN
 U1：DECODER          PORT  MAP(KEYPAD，INNER_KEY)；
 U2：KEY_BUFFER       PORT  MAP(INNER_KEY，  KEY_DOWN，
                                RESET，  INNER_TIME)；
 U3：ALARM_CONTROLLER
                      PORT MAP(KEY_DOWN，ALARM_BUTTON，
                               TIME_BUTTON，CLK，RESET，
                               INNER_L_A，INNER_L_C，
                               INNER_S_N，  INNER_S_A )；
U4：ALARM_COUNTER   PORT MAP( INNER_TIME，  INNER_L_C，
                               INNER_SEC_CLK，  RESET，
                               INNER_TIME_C)；
```

U5：ALARM_REG　PORT MAP(INNER_TIME，INNER_L_A，CLK，
RESET，INNER_TIME_A)；

U6：DISPLAY_DRIVER PORT MAP(INNER_TIME_A，
INNER_TIME_C，INNER_TIME，
INNER_S_N， INNER_S_A，
SOUND_ALARM，DISPLAY)；

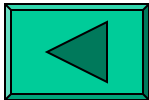
U7：FQ_DIVIDER PORT MAP(CLK，RESET，INNER_SEC_CLK)；

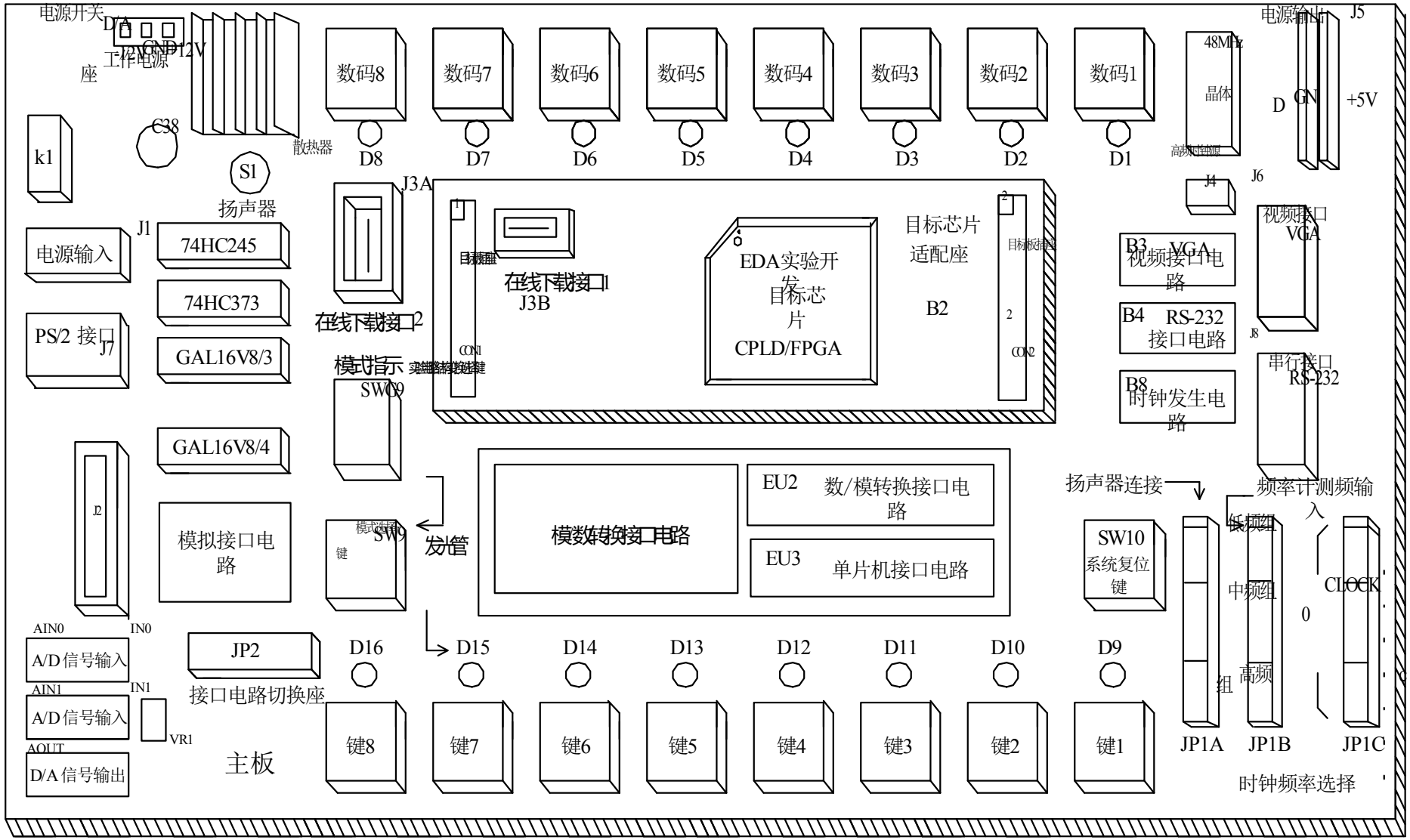
END ART；

（3）闹钟系统的总体仿真结果：

1.1.5 闹钟系统的整体验证

若用GW48型EDA实训开发系统进行硬件验证，考虑到实训开发系统提供的输入信号按键的有限，可将输入数字按键0～9改为一个按键，该按键的信号作为一个8421码信号发生器的输入，由8421码信号发生器输出数字0～9。具体验证方案由大家自行完成。





GW48实训开发系统的板面结构图

基 本 要 求

掌握用 VHDL 语言设计数字电路的一般步骤和设计方法。